NIUF 404-92

North American ISDN Users' Forum Application Software Interface (ASI)

Part 2: MS-DOS Access Method (Version 1)

Approved: June 5, 1992

Application Software Interface Expert Working Group
ISDN Implementors' Workshop
North American ISDN Users' Forum

Revision History

June 1992	Baseline Approved Document (NIUF 404-92)

Abstract

This document describes the Application Software Interface (ASI) access method for the MS-DOS operating system. Although it is a desired goal to have a common ASI for all operating systems, this specification will only apply to the MS-DOS operating system environments. The following restrictions will also apply:

- This access method only applies to a real mode application and will not service any protected mode shells.
- DOS Version 3.2 will be the minimum required DOS revision.
- This issue supports one AE and one PE.

Keywords

application programming interface; API; Application Software Interface; ASI; access method; DOS; implementation agreement; ISDN;

Notice of Disclaimer

Acknowledgments

NIST would like to acknowledge the NIUF Application Software Interface Expert Working Group, and especially the following individuals, for their valuable contributions to this document:

Kenneth A. Argo	ICL
Ron Bhanukitsiri	Digital Equipment Corp.
Cheng T. Chen	Teleos Communications, Inc.
Stephen Halpern	NYNEX Science and Technology
Frank Heath	Rockwell CMC
Dory Leifer	Univ. of Michigan
Jim Loehndorf	Ameritech Services
Ned McCook	DGM&S
Chris Nix	IBM
Stephen Rogers	Electronic Data Systems
Chris Schmandt	MIT Media Lab
Ben Stoltz	Sun Microsystems, Inc.
Robert E. Toense	NIST
Adrian Viego	Bellcore
Wayne Yamamoto	Sun Microsystems, Inc.

Abstract ..... **iii**

Notice of Disclaimer ..... **iv**

Acknowledgments ..... **v**

1.0. Scope **1**

2.0. Overview **1**

3.0. Address Resolution Device Driver **1**

3.1. Overview 1
3.2. Registration and Binding Process 2

4.0. ARDD Message Format **4**

5.0. Callback Function Definitions **4**

5.1. AR Management Function Callback Definition 5
5.2. Management/Control Plane Callback Definition 5
5.3. User Plane Callback Definition 5

6.0. ARDD Commands **5**

6.1. AR-BIND_Mc confirmation [0x01] 6
6.2. AR-BIND_Up confirmation [0x03] 6
6.3. AR-UNBIND_Mc indication [0x05] 6
6.4. AR-UNBIND_Up indication [0x07] 6
6.5. AR-UNBIND_Mc request [0x06] 6
6.6. AR-UNBIND_Up request [0x08] 7
6.7. AR-BIND_Mc request [0x02] 7
6.8. AR-BIND_Up request [0x04] 7
6.9. AR-UNBIND_ALL indication [0x09] 7

7.0. ARDD Access Functions **7**

7.1. Open File with Handle (0x3D) 7
7.2. Close File with Handle (0x3E) 8
7.3. Send Control Data to a Character Device (0x4403) 8
7.4. General IOCTL (0x440C) 9

8.0. Special Commands **10**

8.1. Get Version Number - Minor Code (0x80) 10
8.2. Get ARDD Status Return - Minor Code (0x81) 10

Appendix A: ARDD Status Codes **12**

Appendix B: References **13**

1.0. Scope

This document describes the Application Software Interface (ASI) access method for the MS-DOS operating system. Although it is a desired goal to have a common ASI for all operating systems, this specification will only apply to the MS-DOS operating system environments. The following restrictions will also apply:

- This access method only applies to a real mode application and will not service any protected mode shells.
- DOS Version 3.3 will be the minimum required DOS revision.

2.0. Overview

The ASI access method must support the transfer of information and the thread of execution across the interface. To accomplish these functions and provide for the implementation independence, the MS-DOS access method uses a called interface.

The MS-DOS operating system environment only supports a single thread of execution and a single process (there is no scheduler). **While some special shells exist which provide task switching in the processor's protected mode, their case is not covered by this access method**.

In this environment, the ASI must provide the basic mechanism to transfer the thread of execution across the interface. This mechanism must also be available from either side of the ASI to support the bi-directional asynchronous operation of this interface. The MS-DOS access method, as stated above, will utilize far function calls in both **A**SI **E**ntity (AE) and **P**rogram **E**ntity (PE) as the basic mechanism to transfer the thread of execution. To resolve the addresses of these far functions, the MS-DOS access method has defined a dynamic linking mechanism referred to as the **A**ddress **R**esolution **D**evice **D**river (ARDD). This device driver is defined in detail in the next section. In order to keep ARDD management messages separate from ASI command messages, the AE and PE will have an **A**ddress **R**esolution **M**anagement **F**unction (ARMF) which they will use to receive messages from the ARDD.

ASI functionality requires that the ASI must provide three basic services: management, control, and data. The management and control services, in the MS-DOS access method, will be implemented as a single ASI function. There can only be one **m**anagement/**c**ontrol (MC) function per ASI entity. This is also true for the program entity. The **u**ser **p**lane (UP) service is defined as a separate ASI function. There can be more than one user plane function associated with either the AE or PE.

The information transfer mechanism, across the ASI, will be specific to the access method function. For all management/control plane transfers, the calling entity (AE or PE) will format a message. This message will be passed across the interface by reference (a message pointer). The called ASI access method management/control function will be responsible for transferring the contents of that message at the time of the call. As soon as the called entity return to the calling entity, the message pointer and its contents are invalid.

The user plane function is similar to the management/control plane. Messages will be passed across the interface by reference (a data pointer). The called ASI access method user plane function will be responsible for transferring the contents of that message at the time of the call. As soon as the called entity returns the thread of execution to the calling entity, the message pointer and its contents are invalid.

3.0. Address Resolution Device Driver

3.1. Overview

The **A**ddress **R**esolution **D**evice **D**river (ARDD) is a generic address resolution utility defined for the MS-DOS operating system by the ASI. It can be used to resolve or bind any interface. The ARDD is a character device driver that has these specific characteristics:

1. The ARDD contains multiple device drivers logically linked together. The names shall be "ARDDAE00" and "ARDDPE00".
2. The device attribute word in the device header shall be coded such that the device is a character device that will support "open", "close", "IOCTL Read", and "IOCTL Write" calls. This value should be 0xC840.
3. Information is passed to the device driver via the IOCTL Write command, and returned by the device driver through a call to the AR Management Function callback (see section 3.2 "Registration and Binding Process" for more details).
4. Initialization, Open, Close, IOCTL Write, and Generic IOCTL are the only valid commands that this device driver will process.
5. The ARDD must be the first device driver installed in the "config.sys" file driver list relative to any device driver that may utilize the ARDD.
6. The ARDD will respond to the Generic IOCTL, Category = ARDD (0x0A), Minor Code = "Get Version Number (0x80)", with a Major value of 0x10 and Minor value of 0x0.
7. The ARDD will respond to the Generic IOCTL, Category = ARDD (0x0A), Minor Code = "Get ARDD Status Return (0x81)", with the status of the last error.

Beyond these requirements, there are no sequence related dependencies imposed upon the user. This means that no restrictions are placed on the AE and PE with regards to which one loads or binds first.

The ARDD uses a simple command interface. The valid commands are:

• AR-BIND_Mc confirmation	0x01
• AR-BIND_Mc request	0x02
• AR-BIND_Up confirmation	0x03
• AR-BIND_Up request	0x04
• AR-UNBIND_Mc indication	0x05
• AR-UNBIND_Mc request	0x06
• AR-UNBIND_Up indication	0x07
• AR-UNBIND_Up request	0x08
• AR-UNBIND_ALL request	0x09

NOTE: There are no "response" primitives in this version of the DOS access method.

3.2. Registration and Binding Process

To provide symmetry with the rest of the ASI, the ARDD binding procedure is asynchronous. In order to provide this asynchronous operation the request and confirm operations are non-symmetrical. The request operation is carried out via an IOCTL **"Send Control Data to a Character Device"** (INT 21, function 0x4403) which results in a call to the device driver's IOCTL Write entry point (remember, the calling program must open the device driver first). The confirm operation is carried out when the ARDD calls the AR Management Function (ARMF) callback of the AE or PE which was supplied during the request operation.

It is important to note that while there is only one ARDD, it has two entry points. This allows the ARDD to more easily differentiate between two entities which need to be bound since each entity will always call upon a specific entry point. When a bind request arrives at each entry point carrying the same id, the ARDD notifies both entities of the resulting match. Also, by having two entry points which **always remain open** (while the calling entity is active), the ARDD is capable of detecting when a process goes away prematurely and can send AR_UNBIND_ALL indications to the other entity to provide for graceful cleanups.

The following scenario represents a typical load and address resolution procedure. While this scenario is not specific about each step that an entity performs or the time frame in which an action occurs, it is assumed that all entities are well behaved relative to their environment and follow proper procedural practices. This scenario makes the assumption that the AE was loaded first. With that assumption, the sequence of these events must be observed. This scenario also assumes no errors (either logical or physical) occur.

While this scenario is typical, it in no way implies that it is the only valid sequence which can be used. Steps 9 - 15, for instance, can easily be changed to present a scenario where the PE initiates the User Plane binding.

1. The Address Resolution Device Driver (ARDD) loads and initializes.
2. The ASI Entity loads. It then opens the ARDD using the name "ARDDAE00". Using the device handle returned on the open, the AE formats an AR-BIND_Mc request and issues it to the ARDD via the IOCTL "Send Control Data to a Character Device" command. This request contains the AE's MC plane function address and the AE's Address Resolution Management Function (ARMF) address.

   Note: Do not close the ARDD.
3. The ARDD registers the MC and ARMF addresses and returns an 0xA1 in the AL register with the Carry Flag Set.

3a. The AE issues the IOCTL "Generic IOCTL" with the category set to ARDD [0x0A] and the minor code set to "Get ARDD Status Return" [0x81]. The code returned in the status word will be 0x00A1 (registered).

4. The Program Entity loads. It then opens the ARDD using the name "ARDDPE00". Using the device handle returned on the open, the program entity formats an AR-BIND_Mc request and issues it to the ARDD via the IOCTL "Send Control Data to a Character Device". This request contains the PE's MC plane function address and the PE's Address Resolution Management Function (ARMF) address.

Steps 5 - 8 occur while the ARDD is servicing the interrupt routine which resulted from the IOCTL call in step 4 .

5. The ARDD registers the MC and ARMF addresses of the PE.
6. The ARDD calls the PE's ARMF callback with an AR-BIND_Mc confirmation, passing the AE's MC function address. The callback returns 0x00 to the ARDD.
7. The ARDD calls the AE's ARMF callback with an AR-BIND_Mc confirmation, passing the PE's MC function address. The callback returns 0x00 to the ARDD.
8. The ARDD returns a 0x00 to the PE (This completes the IOCTL call in step 4).

At this point the PE and AE are bound. The PE can initiate a call to the AE's management/control plane function. Via this function, configuration of the ASI entity can take place (see fig. 3 in this document).

9. The PE calls the ARDD with an AR-BIND_Up request passing the user plane and management/control plane function (far) addresses (Reference_Id = 0x00000005). The ARDD returns 0xA1 to the PE.
10. The PE calls the AE's control plane function with an Nb-CONNECT request (PEI = 0x0005).
11. The AE calls the ARDD with an AR-BIND_Up request, passing the user plane data, and control function (far) addresses (Reference_Id = 0x00060005).
12. The ARDD calls the PE with an AR-BIND_Up confirmation passing the AE's user plane function (far) address (Reference_Id = 0x00060005).
13. The ARDD calls the AE with an AR-BIND_Up confirmation passing the PE's user plane function (far) address (Reference_Id = 0x00060005).

14. Here, we assume that the appropriate processing goes on in the AE, and an ISDN SETUP is sent to the network.
15. The ARDD returns 0x00 to the AE.
16. A CONNECT is received from the network, which then generates an interrupt to transfer the thread of execution to the AE's interrupt handler.
17. The AE calls the PE control plane function with an Nb-CONNECT confirmation (AEI = 0x0006, PEI = 0x0005). The PE returns 0x00.
18. The AE issues an interrupt return (IRET) and transfers the thread of execution back to the PE.

Now both the PE and AE can effect data transfers by calling the other's data and control function.

The control function address provides a mechanism for data-synchronous user plane control information. The use of this information will be specific to the peer-to-peer data protocol employed across the ISDN connection. The format of the user plane control messages is for further study.

Note: The PE's callback routine should only queue messages and return immediately so that the thread of execution can travel back and forth effectively. Once the AE returns control to the PE, the PE can the check the messages which have been posted, and act on them accordingly.

4.0. ARDD Message Format

Messages which are sent to the ARDD via the IOCTL "Send Control Data to a Character Device" command or received from the ARDD via the AR Management Function have the following format.

Byte(s)	Contents
1	Command
2-5	Reference_Id
6-9	Callback_Address_1
10-13	Callback_Address_2

Note: Referenece_Id contains a copy of the four octets, 2 through 5, as defined in Part 1, Commands chapter, Message Format section.

5.0. Callback Function Definitions

This ASI access method defines three types of callback routines:

ARMF	**AR**DD **M**anagement **F**unction
MC	**M**anagement / **C**ontrol plane
UP	**U**ser **P**lane

These callback routines provide for the asynchronous transfer of information typically in the Up Stream (going from an entity which is logically lower in the architecture to one which is logically higher) direction. The exception is the ARDD's ARMF callback which is used by the ARDD to provide indications and confirmations to both the AE and PE.

This section provides the calling sequences for the various callback functions. All of the callbacks provide the same return codes as defined below.

If successful, the CARRY flag will be cleared and the AX register will be set to 0. If the transfer fails, the CARRY flag will be set and AX will contain one of the error codes found in Appendix A.

NOTE: The callback is responsible for preserving all registers except AX; the caller is responsible for maintaining stack integrity. Error codes are returned in AX.

5.1. AR Management Function Callback Definition

The AR Management Function callback transfers the thread of execution and the message from the ARDD to the called entity. The following calling procedure will be used:

```
PUSH WORD      MsgSize            Bytes in Data Message
PUSH DWORD     MsgPtr             Pointer to Data Message (ARDD Formatted buffer)
CALL           AR_Callback
```

5.2. Management/Control Plane Callback Definition

The Management/Control Plane callback transfers the thread of execution and the message from the calling entity to the called entity (i.e., PE -> AE or AE -> PE). The following calling procedure will be used:

```
PUSH WORD      MsgSize            Bytes in Data Message
PUSH DWORD     MsgPtr             Pointer to Data Message (ASI Formatted buffer see Command
                                  section in Part 1)
CALL           Mc_Callback
```

5.3. User Plane Callback Definition

The User Plane callback transfers the thread of execution and the message from the calling entity to the called entity (i.e., PE -> AE or AE -> PE). The following calling procedure will be used:

```
PUSH WORD      DataSize           Bytes in Data Message
PUSH DWORD     DataPtr            Pointer to Data Message (Protocol Specific)
PUSH WORD      CtlSize            Bytes in Control Message
PUSH DWORD     CtlPtr             Pointer to Control Message (Protocol Specific)
CALL           Up_Callback
```

Note: The Control Message may not be required for most data transfers. The Control Message should be used when a protocol requires the passing of signaling information to upper layers which is synchronized with a data message. When a control message is not present, the **CtlSize** should be set to zero and **CtlPtr** is invalid. CtlSize being set to zero informs the receiving entity to ignore CtlPtr. When a data message is not present, the **DataSize** should be set to zero and the **DataPtr** is invalid. DataSize being set to zero informs the receiving entity to ignore DataPtr.

6.0. ARDD Commands

This section contains the descriptions of the messages associated with each ARDD command. When reading this chapter, the following criteria applies:

Each section title contains the **Command Name** in bold letters followed by an indicator which denotes the direction of the command (request or confirmation).

There are four types of indicators which are possible:

request	requests are sent from the AE or PE to the ARDD to specify a BIND or UNBIND procedure.
confirmation	confirmations are sent from the ARDD to the AE or PE to specify the completion of a BIND or UNBIND procedure.

indication	indications are sent from the ARDD to the AE or PE to inform them of an unsolicited event (i.e., an AR-UNBIND_Up. indication is sent to the PE if the AE sends an AR-UNBIND_Up request to the ARDD).
response	responses are sent from the AE or PE to the ARDD to acknowledge the reception of an indication. **NOTE: This version of the DOS access method does not issue responses.**.

Following each title is a brief description of the command and a listing of the required parameters. Values for these parameters are not specified. Each title ends with the actual hex value of the command.

6.1. **AR-BIND_Mc** confirmation **[0x01]**

Provides the called entity with an acknowledgment that its management plane has been bound.

```
Reference_Id   4 bytes           // set = 0
Mc_Callback    far pointer       // Management / Control plane callback
```

Return:

See Appendix A

6.2. **AR-BIND_Up** confirmation **[0x03]**

Provides the called entity with an acknowledgment that its user plane has been bound.

```
Reference_Id   4 bytes           // set = AEI / PEI pair
Up_Callback    far pointer       // User Plane callback
```

Return:

See Appendix A

6.3. **AR-UNBIND_Mc** indication **[0x05]**

Provides the called entity with an indication that the entity with which its management plane was bound is no longer available.

```
Reference_Id   4 bytes           // set = 0
```

Return:

See Appendix A

6.4. **AR-UNBIND_Up** indication **[0x07]**

Provides the called entity with an indication that the entity with which its user plane was bound is no longer available.

```
Reference_Id   4 bytes           // set = AEI / PEI pair
```

Return:

See Appendix A

6.5. **AR-UNBIND_Mc** request **[0x06]**

Instructs the ARDD to remove its management plane entry from the appropriate list and notify the entity to which it was bound that it is no longer available.

```
Reference_Id   4 bytes           // set = 0
```

Return:

See Appendix A

6.6. **AR-UNBIND_Up** request **[0x08]**

Instructs the ARDD to remove its user plane entries from the appropriate list and notify the entity to which it was bound that it is no longer available.

```
Reference_Id   4 bytes           // set = AEI / PEI pair
```

Return:

See Appendix A

6.7. **AR-BIND_Mc** request **[0x02]**

Instructs the ARDD to bind its management plane.

```
Reference_Id   4 bytes           // set = 0
AR_Callback    far pointer
Mc_Callback    far pointer
```

Return:

See Appendix A

6.8. **AR-BIND_Up** request **[0x04]**

Instructs the ARDD to bind its user plane.

```
Reference_Id   4 bytes
AR_Callback    far pointer
Up_Callback    far pointer
```

Return:

See Appendix A

6.9. **AR-UNBIND_ALL** indication **[0x09]**

Instructs the ARDD to unbind all connections.

```
Reference_Id   4 bytes           // set = 0
```

Return:

```
0x0000                  OK                // Operation successful
```

7.0. ARDD Access Functions

The following sections provide examples of the DOS functions used to communicate with the ARDD. For further information, see Microsoft's MS-DOS Programmer's Reference (Document Number SY0766b-R50-0691).

All of the functions in this section use services provided by interrupt 21h (33).

7.1. Open File with Handle (0x3D)

To open the ARDD, place the device name "ARDDAE00" or "ARDDPE00" in a buffer and call DOS with a pointer to that buffer.

Example:

```
mov            dx, seg FileName   ; get segment of buffer
mov            ds, dx             ;   and place it in ds
mov            dx, offset FileName; ds:dx points to device name
mov            al, 12h            ; mode = OPEN_ACCESS_READWRITE | OPEN_SHARE_DENYREADWRITE
mov             ah, 3Dh            ; "Open File with Handle"
int             21h                ; call DOS for service
jc              error_handler      ; if "CARRY" set, we had an   error
mov             Handle, ax         ; store handle for later
                .
                .
                .

FileName        DB 'ARDDAE00',0    ; file name buffer
Handle          DW ?               ; handle returned from open
```

Return:

If "CARRY" flag clear:

```
AX = Handle of open device.
```

If "CARRY" flag set, AX may contain one of the following errors:

```
0002h                   File not found
0003h                   Path not found
0004h                   Too many open files
0005h                   Access Denied
000Ch                   Invalid Access
```

7.2. Close File with Handle (0x3E)

To close the ARDD, provide the handle which was returned from the open.

Example:

```
mov             bx, Handle         ; handle of device
mov             ah, 3Eh            ; "Close File with Handle"
int             21h                ; call DOS for service
jc              error_handler      ; if "CARRY" set, we had an error
                .
                .
                .
                .
Handle          DW ?               ; handle returned from open
```

Return:

If "CARRY" flag clear:

```
function completed
```

If "CARRY" flag set, AX may contain the following error:

```
0006h                   Invalid Handle
```

7.3. Send Control Data to a Character Device (0x4403)

To send commands to the ARDD, use the handle obtained from the open.

Example:

Assume that "Buffer" is a buffer containing a properly formatted ARDD message. "Handle" is a word which received the result of the open. "MaxBytes" is a word which contains the actual number of bytes in "Buffer". "ActualBytes" is a word which receives the results of the IOCTL.

```
mov             bx, Handle         ; handle of device
mov             cx, MaxBytes       ; # of bytes to send
mov             dx, seg Buffer     ; get segment of buffer
mov            ds, dx
mov            dx, offset Buffer  ; ds:dx points to data
mov            ax 4403h           ; "Send Control Data to a
                                  ;   Character Device"
int            21h                ; call DOS for service
jc             error_handler      ; if "CARRY" set, we had an
                                  ;   error
mov            ActualBytes, ax    ; number of bytes sent
               .
               .
               .
               .
Handle         DW ?               ; handle returned from open
ActualBytes    DW ?               ; holds number of bytes accepted by ARDD

Buffer         DB MaxBytes dup (? ; the actual number of bytes required by your
                                  ; application may differ
```

Return:

If "CARRY" flag clear:

```
function completed
```

If "CARRY" flag set, AX may contain one of the following errors:

0001h	Invalid Function
0005h	Access Denied
0006h	Invalid Handle
000DH	Invalid Data

7.4. General IOCTL (0x440C)

To send commands to the ARDD, use the handle obtained from the open.

Example:

Assume that "Buffer" is a buffer containing information to be passed to the ARDD or a buffer to receive information from the ARDD. "Handle" is a word which received the result of the open. "MajorCode" and "MinorCode" represent the command ID for the ARDD function you wish to have performed.

Note: The buffer must be large enough to hold any information returned by the ARDD.

```
mov            bx, Handle         ; handle of device
mov            ch, MajorCode      ; major function code
mov            cl, MinorCode      ; minor function code
mov            dx, seg Buffer     ; get segment of buffer
mov            ds, dx
mov            dx, offset Buffer  ; ds:dx points to data
mov            ax 440Ch           ; "General IOCTL"
int            21h                ; call DOS for service
jc             error_handler      ; if "CARRY" set, we had an error
               .
               .
               .
               .
Handle         DW ?               ; handle returned from open
Buffer         DB 200 dup (?)     ; the actual number of bytes required by each call
                                  ; may differ.
```

Return:

If "CARRY" flag clear:

```
function completed
```

If "CARRY" flag set, AX may contain one of the following errors:

```
0001h                 Invalid Function
0005h                 Access Denied
0006h                 Invalid Handle
000DH                 Invalid Data
```

8.0. Special Commands

This version of the DOS access method defines a special category and two minor codes for use with the IOCTL "Generic IOCTL". The special category is (**0x0A**) and is used to denote a "Generic IOCTL" meant only for the ARDD. The minor codes represent special commands which the ARDD must support through this IOCTL. This section describes those commands.

8.1. Get Version Number - Minor Code (0x80)

Get Version Number is used by the AE and PE to get the version number of the DOS access method supported by the ARDD. The calling entity passes the address of a buffer containing the buffer length in the first word. On return, the first word contains the actual number of bytes filled in. The structure of the buffer is as follows:

Bytes	Description
1-2	Length of buffer
3	Major Number
4	Minor Number
5-12	Vendor String
13	Vendor Major Number
14	Vendor Minor Number

```
mov          ax, 0Eh            ; Length of buffer
mov          Buffer, ax         ; set the buffer length in the first byte
mov          bx, Handle         ; handle returned by DOS open
mov          ch, 0Ah            ; major function code (ARDD)
mov          cl, 80h            ; minor function code (Get Version Number)
mov          dx, seg Buffer     ; get segment of buffer
mov          ds, dx
mov          dx, offset Buffer  ; ds:dx points to data
mov          ax 440Ch           ; "General IOCTL"
int          21h                ; call DOS for service
jc           error_handler      ; if "CARRY" set, we had an error
             .
             .
             .
             .
Handle       DW ?               ; handle returned from open
Buffer       DB 14 DUP (?)      ; Version Number [Version, Release, etc...]
```

8.2. Get ARDD Status Return - Minor Code (0x81)

Get ARDD Status Return allows the AE or PE to obtain the status message which indicates the ASI error which occurred as a result of an AR-BIND_xx or AR-UNBIND_xx request. This call is only used when a request to the ARDD by the AE or PE returns with the Carry Flag set. See Appendix A for possible values.

```
mov            bx, Handle         ; handle returned by DOS open
mov            ch, 0Ah            ; major function code (ARDD)
mov            cl, 81h            ; minor function code (Get ARDD Status Return)
mov            dx, seg Buffer     ; get segment of buffer
mov            ds, dx
mov            dx, offset Buffer  ; ds:dx points to data
mov            ax 440Ch           ; "General IOCTL"
int            21h                ; call DOS for service
jc             error_handler      ; if "CARRY" set, we had an error
               .
               .
               .
               .
Handle         DW ?               ; handle returned from open
Buffer         DW ?               ; Status Word
```

Appendix A: ARDD Status Codes

This section contains the definitions of the possible status returned by the ARDD after an unsuccessful bind or unbind request.

Value	Description
0x81	Message length error
0x82	Invalid message pointer
0x83	Out of memory
0x84	Function not supported
0x85	Interface Busy

Value	Description
0x0000	OK
0x00A1	Registered
0x00A2	Invalid Reference
0x00A3	Reference Not Unique
0x00A4	Active Register / Bind
0x00A5	Not Bound

Appendix B: References

2.1 ANS Documents

[1] ANS T1.607-1990, *Telecommunications — Integrated Services Digital Network (ISDN) — Digital Subscriber Signalling System Number 1 (DSS1) — Layer 3 Signalling Specification for Circuit-Switched Bearer Service*.

2.2 CCITT Documents

[2] CCITT Recommendation I.320 - 1988, *ISDN Protocol Reference Model*.

[3] CCITT Recommendation I.515 - 1988, *Parameter Exchange for ISDN Networking*.

[4] CCITT Recommendation Q.921-1988 (also designated CCITT Recommendation I.441-1988), *ISDN User-Network Interface Data Link Layer Specification*.

[5] CCITT Recommendation Q.931-1988 (also designated CCITT Recommendation I.451-1988), *ISDN User-Network Interface — Layer 3 Specification for Basic Call Control*.

[6] CCITT Recommendation V.110 -1988, *Support of Data Terminal Equipments (DTEs) with V-series Type Interfaces by an Integrated Services Digital Network (ISDN)*.

[7] CCITT Recommendation V.120 -1988, *Support by an ISDN of Data Terminal Equipment with V-series Type Interfaces with Provision for Statistical Multiplexing*.

[8] CCITT Recommendation X.25 -1984, *Interface between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) for Terminals Operating in the Packet Mode and Connected to Public Data Networks by Dedicated Circuit*.

2.3 ISO Documents

[9] ISO 8824:1987(E), *Information processing systems — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1)*.

[10] ISO 8825:1987(E), *Information processing systems — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*.

2.4 Other Documents

[11] NIST Special Publication 500-183, *Stable Implementation Agreements for Open Systems Interconnection Protocols*, Version 4, Edition 1, December 1990.

[12] *MS-DOS Programmer's Reference* (Document Number SY0766b-R50-0691).